Bellman Ford Algorithm

This works for negative edge weights as well. However, we always need a directed graph. In case, we do not have directed graph we can add the same edge twice with source & destination flipped.

It is single source starting point algorithm.

The order of edges is irrelevant for this algorithm. However, it requires all edges with their weights.

Relax all the edges N-1 times sequentially.
Relaxing means if distance to current node (dist[node]) plus weight of edges is less than distance (dist[dest]) we update dist[dest]. The dist[node] should be non-infinity to perform this operation.

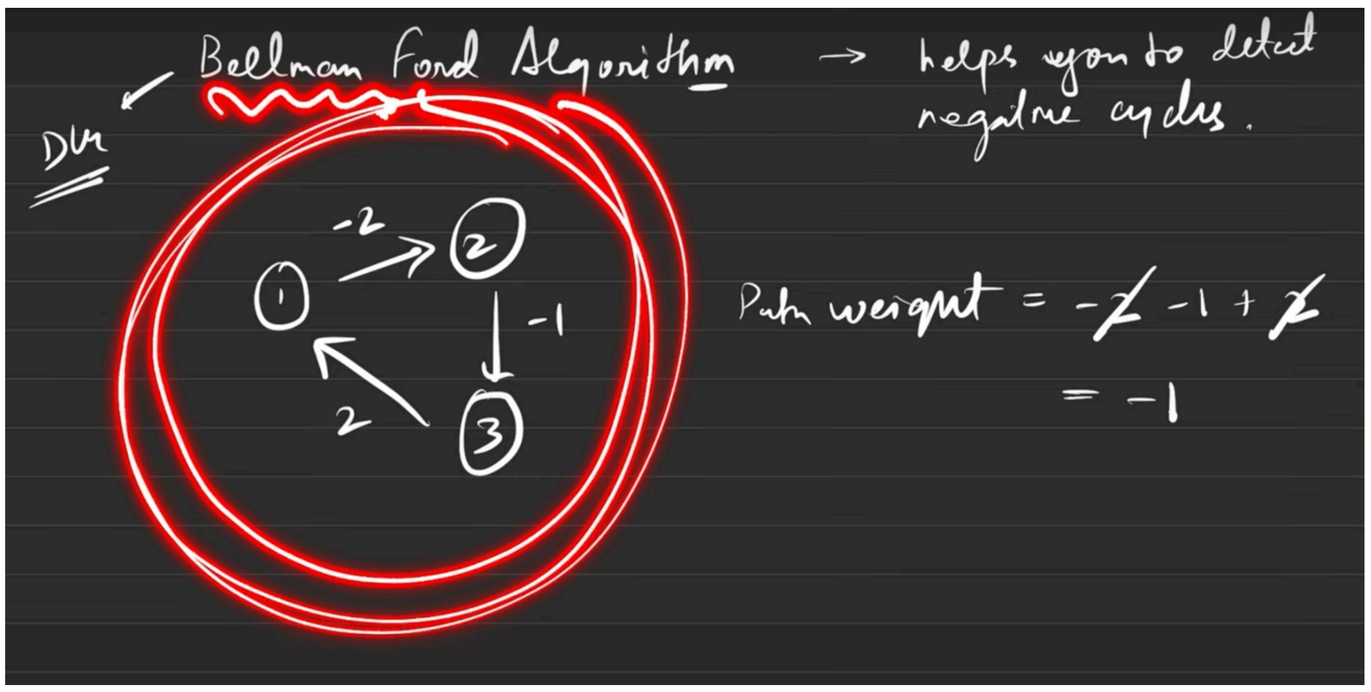
Initially distance array will have all nodes distance marked as inf except source which will be marked as zero.

Questions -

Why relax n-1 times?
It's because in worst case scenario we will only get one new value calculated from any point in graph and starting node has zero distance anyways so we don't need to consider this.

How to know if we have -ve cycle?
If even after n-1 iterations on $n^{th}$ iteration we are still reducing a distance, it's negative cycle.

Network Delay Time is a good practice problem for this algo & Dijkstra's.




```
/*
*   edges: vector of vectors which represents the graph
*   S: source vertex to start traversing graph with
*   V: number of vertices
*/
class Solution {
    static int[] bellman_ford(int n, ArrayList<ArrayList<Integer>> edges, int S) {
        // Write your code here
        int[] dist = new int[n];
        Arrays.fill(dist, (int) 1e8 );
        dist[S] = 0;

        for (int i = 0; i < n; i++) {
            for (ArrayList<Integer> edge: edges) {
                int u = edge.get(0), v = edge.get(1), wt = edge.get(2);
                if (dist[u] != (int) 1e8 && dist[u] + wt < dist[v]) {
                    dist[v] = dist[u] + wt;
                }
            }
        }

        for (ArrayList<Integer> edge: edges) {
            int u = edge.get(0), v = edge.get(1), wt = edge.get(2);
            if (dist[u] != (int) 1e8 && dist[u] + wt < dist[v]) {
                return new int[] { -1 };
            }
        }


        return dist;
    }
}
```